

Design and Implementation of a Scalable Intrusion Detection System for the Protection of Network Infrastructure *

Y. F. Jou, F. Gong, C. Sargor, X. Wu
Advanced Networking Research
MCNC
RTP, NC 27709

S. F. Wu, H.C. Chang, F. Wang
Computer Science Dept.
NC State University
Raleigh, NC 27695

Abstract

This paper presents the design, implementation, and experimentation of the JiNao intrusion detection system (IDS) which focuses on the protection of the network routing infrastructure. We used Open Shortest Path First (OSPF) routing protocol as an implementation example to illustrate our IDS design. However, the system architecture is generic enough that the JiNao IDS can be used for protecting other protocols. The system features attack prevention and intrusion detection with tightly integrated network management components. The prevention module functions like a firewall which consists of a small set of rules. Both misuse (protocol analysis) and anomaly (statistical based) approaches are implemented as detection mechanisms in order to handle both known and unknown attacks. Four OSPF attacks (i.e., MaxSeq, MaxAge, Seq++, and LSID attacks) have been developed for evaluating JiNao's detecting capability. Furthermore, an SNMP based network management interface has been designed and implemented such that the JiNao IDS can be easily integrated with existing network management systems.

1 Introduction

Given the recent growth of the Internet, intrusion incidents are becoming common events of life. Some of these incidents are simply out of innocuous curiosity. Some, however, are due to malicious attempts in order to compromise the availability of the information system or the integrity and privacy of the information itself. Despite the best efforts of the protocol designers, implementors, and system administrators, it is prudent to assume that attacks will occur and some, unfortunately, will succeed. Therefore, it is vitally important to develop means to automatically detect and respond to these attacks in real-time in order to main-

tain critical information services. Since routing protocols (e.g., RIP, EIGRP, OSPF, BGP, and MPLS) form the very heart of the Internet infrastructure, the security issues regarding IP routing is of the utmost importance.

Routing protocols can be classified as either *distance-vector* or *link-state*. In distance-vector based protocols, nodes keep tables of the best paths and associated metrics for all possible destinations and periodically exchange the contents of the tables with their neighbors. That is, a router will tell all its neighbors about its connections to the whole world. On the other hand, link state protocols are characterized by every node keeping a "map" of the entire network which is used to compute shortest paths to all destinations. Each node contributes to this global view by periodically distributing (via flooding) link state advertisements (LSAs). It has been identified in [1, 2] that link state routing protocols, like OSPF [5], are more robust against simple failures or attacks. In this paper, we limit our focus to link-state routing protocols only.

Murphy and Badger from TIS [6] proposed a preventive approach by using public key signature scheme to protect the integrity of LSAs (Link-State Advertisement) in OSPF. With a public key infrastructure, the source router uses its private key to sign the MAC (Message Authentication Code) for every LSA created. Since the intermediate routers do not know the private key of the source router, they can not tamper with the LSAs without being detected. Each receiver of LSAs must use the source router's public key to verify its integrity. Therefore, their scheme is secure against compromised intermediate routers. Unfortunately, this proposal has been voted down by the OSPF working group within IETF because (1) the public key operations involved are too expensive, and (2) the TIS version of OSPF protocol is more complex than the original version.

JiNao, a joint research project between MCNC and

*This work is supported in part by DARPA/ITO through Contract # F30602-96-C0325

North Carolina State University, proposed a comprehensive approach which features both attack prevention and intrusion detection to handle attacks for link-state routing protocol. Since the detection process does not require any modifications to the OSPF protocol engine, this is a very practical approach to handle attacks as it does not need to go through the lengthy IETF standardization process.

In this paper, we present the JiNao intrusion detection system with the OSPF routing protocol as an example for protection. Section 2 provides an overview of the OSPF routing protocol. The JiNao system architecture is presented in Section 3. Sections 4 and 5 describe in detail the protocol and statistical analysis modules. Finally, in Section 6, we discuss our experience in using the JiNao system in detecting routing protocol attacks.

2 Link State Routing and OSPF

2.1 Dynamic Route Update

Link state routing protocols such as OSPF create a global network topology map in three distinct phases:

Adjacency Establishment: An OSPF router sends *Hello (or Keep-Alive)* packets periodically to discover its neighboring routers. Three attributes are included in each Hello packet: *hello interval* indicates the frequency for sending out such Hello packets, *router dead interval* specifies the time it takes to declare a router unavailable, and *neighbor list* describe the list of neighbors that the sender has already met. Once neighboring routers have “met” via the Hello Protocol, they go through a Database Exchange Process to synchronize their databases with one another.

Information Sharing by LSA Flooding: The link-state information about a router’s local neighborhood is assembled into a *Link-State Advertisement (LSA)*, and *flooded* throughout the whole network to reach all other routers.

Shortest Route Path Calculation: Once a router collects all the link-state information, it uses the Dijkstra algorithm to calculate a shortest path tree with the router itself as the root node and then forms a complete picture of routing in the network.

2.2 Hierarchical Routing

In order to reduce routing traffic and the size of the topology database, OSPF adopted a two-level hierarchical routing scheme, *area* and *backbone*, within an *Autonomous System (AS)* in which a collection of computing systems, routers and other network devices share a single administrative entity.

In OSPF, an *area* is a collection of networks, hosts, and routers connecting each other. Each area runs a separate copy of the basic link-state routing algorithm. This means that each area has its own link-state database. The topology details of an area are hidden from the outside of the area. Conversely, routers internal to a given area know nothing about the detailed topology external to the area. This isolation helps not only to reduce routing traffic but also to constraint the bad effect of malicious routing protocol attacks being launched from a different area. Furthermore, each autonomous system has one special area called *Backbone*, which serves as the hub of this AS and all areas in the same AS must connect to this backbone.

2.3 OSPF Packet Types

OSPF defines five types of control packets. *Hello* packets are used by two adjacent routers to maintain a neighborhood relationship. *Database Description* and *Link State Request* packets are used to synchronize two router’s database when an adjacency is being initialized. *Link State Update* and *Link State Acknowledge* packets are used to broadcast LSAs (Link State Advertisement), and will be explained in detail in the next section.

2.4 Link-State Advertisement (LSA)

A link state update OSPF packet carries one or more LSA (Link-State Advertisement) instances describing the current status of one or more network links. The header of an LSA instance is shown in Figure 1. *LSA Age* (the first 16 bits in the header) is set to zero by the originator, and is incremented on every hop during the flooding procedure. The age of an LSA is also incremented as it is held in each router’s topology database until it reaches its maximum value, 3,600 (one hour). A *maxage* LSA instance is considered out of date, and should be purged out of a router’s database. If an originator decides to purge an out-of-date LSA instance out of its database, it should re-flood this *maxage* LSA as a signal for other routers to also remove the same LSA instance from their database. This guarantees the consistency among the distributed network topology database.

Here, a distinction must be made between an *LSA* and an *LSA instance*. An *LSA* is associated with a particular link or network. For example, let’s suppose that there is a link connecting router *A* and *B*. Router *A* is responsible for originating an *LSA* to tell other routers that it has a link to router *B*, while router *B* will use another *LSA* to tell others that it has a link to router *A*. An *LSA instance* gives the state of

0	16	31
LS age	Option	LS type
Link State ID		
Advertising Router ID		
LS sequence number		
LS checksum	length	

Figure 1: LSA header format

a particular LSA at a particular time. For example, router *A* at time t_1 may broadcast an *LSA instance* saying the cost for its link to router *B* is 10. After a while, say, at time t_2 , the status of the link changes. Router *A* will broadcast a new *LSA instance* telling the new cost for this link. In a network, there may exist more than one instance for a particular LSA.

The *LS sequence number* field in the LSA header, a signed 32-bit integer, is used as an indicator to compare the freshness of two different LSA instances for the same LSA. The smallest LS sequence number is 0x80000001, and the largest number is 0x7fffffff, while 0x80000000 is reserved by OSPF. For instance, an LSA instance with seq# 0x91000123 is considered to be newer than another instance 0x87234006. The originator of an LSA will increment the sequence number by 1 for each new LSA instance and other routers will accept the newer/updated link state information being carried by the new LSA instance. When an attempt is made to increment the maximum sequence number, i.e., 0x7fffffff, the MaxSeq LSA instance must be flushed from the routing domain first. This is achieved by prematurely aging the MaxSeq LSA to MaxAge (3,600) and re-flooding it through the network. After the purging, a new instance can then be originated with sequence number 0x80000001.

2.5 OSPF Security

Two inherent mechanisms of OSPF make it very robust and resilient to failures, even to some malicious attacks.

Flooding and information least dependency: As we mentioned before, LSAs are propagated by flooding; the flooding algorithm is reliable, which ensures all routers in the same area have the same topological database. Consider either a single point (router) failure case or an intruder trying to fake or modify other router's information: As long as there is another alternate path, all good routers will finally receive both original LSA and the corrupted LSA. This

triggers an interesting phenomenon in OSPF *fight-back*: a good router trying to *convince* a bad router by continuing to send it correct information, as observed in our previous work [9]. We think it is an advantage in that it could be easily spotted by an *Intrusion Detection System* (IDS).

A more profound impact of flooding individual LSA is *information least dependency*: every router uses the *raw information* from the original advertiser instead of *aggregated information* from neighbors, which gives security advantages over other pure distance vector based routing protocols.

In a distance vector algorithm (e.g., RIP), each router sends only summarized information, which are computational results based on reachability information from its neighbors. This aggregation of information has two implications. First, it is very hard for a router to validate the information it receives. Second, even if a router detects incorrect information, it is still difficult to determine the source of the corruption.

By comparison, in a link state routing algorithm such as OSPF, each router generates information about its local topology (e.g., its neighbors), and also forwards such information to other routers via flooding. This has several advantages: every router independently possesses the entire topology information for the network and each router is responsible only for its own local portion of the topology. As long as at least one of its neighbor is honest, it can get raw independent information of the whole world. Obviously, information independence helps (compared to distance vector) to find out which router is corrupted. Also, independence makes it possible to use authentication to verify the origin of a message. Recently, the method proposed in [7, 8] which uses predecessor-based information to harden distance vector algorithms, justifies the principle from another viewpoint: a predecessor is essentially a piece of information provided by source to alleviate the total blindness of router. By going through the predecessor, a router can help reconstruct the shortest path tree back to the source. It is the predecessor information that makes a secure distance-vector based algorithm possible.

Hierarchy routing and information hiding: The primary goal of hierarchical routing is to deal with routing scalability issues (reduce routing table size, link bandwidth and router computing resources). But, we also see it has both robustness and security advantages. OSPF is a two-level routing protocol: intra-area and inter-area routing, with ABR (area border router) connected to backbone and exchanging area summary information. There are three cases we can

consider here:

- *Internal router is compromised:* The implication of two-level routing is that an internal router does not need to know the topology of the outside except its own area. Consider the case where an internal router is compromised. The damage it can do is very much limited to its area, without impacting the routing in other areas.
- *ABR (area border router) is compromised:* If there is only one ABR or this ABR is the only one attached to the backbone, then the area will suffer serious consequences. If there is other ABRs online, since all ABRs for an area should broadcast the same topology information, redundancy would provide connectivity and mutual verification to possibly detect any conflict of information.
- *ASBR (autonomous system boundary router) is compromised:* OSPF uses ASBR to import external routing information into OSPF routing domain. These external routing information will be flooded through the routing domain. What the ASBR does is actually punch a *hole* in the area boundary. It is probably the single worst security vulnerability in OSPF and there is no easy way to fix this. Database overflow protection can prevent an intruder from arbitrarily flooding junk routes to a certain degree, but is incapable of detecting false or fake routes.

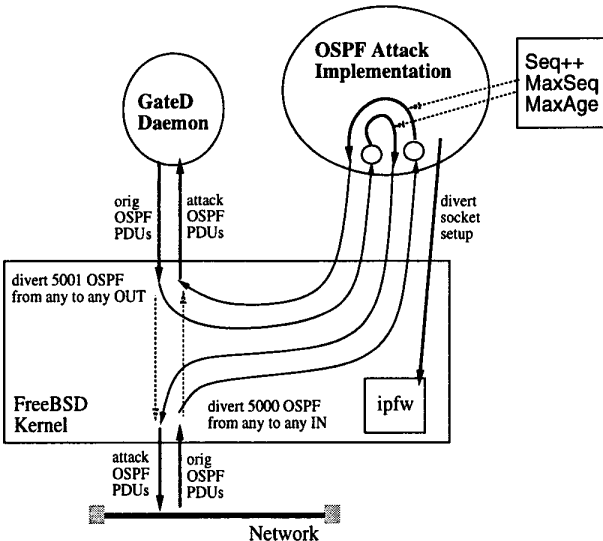


Figure 2: OSPF Attack Implementation

2.6 OSPF Attacks

In order to validate the JiNao IDS system, four OSPF attacks [9] have been implemented for both the FreeBSD and Linux platforms through a mechanism called *divert socket*. Divert sockets enable both IP packet interception and injection on the end-hosts as well as on the routers. Interception and injection happen at the IP layer. The intercepted packets are diverted to sockets in the user space, thus they will not be able to reach their destination unless they are reinjected by the user space sockets. This allows different tricks (e.g., routing and firewall) to be played, outside the operating system kernel, in between the packet interception and reinjection. The divert socket on Linux was implemented by MCNC [10]. The attacks themselves are implemented in the user process, and the tampered OSPF packets are then re-injected into the kernel, which will deliver these packets to either the routing daemon running on the same machine (incoming direction) or its neighbors (outgoing direction). The architecture of attack implementation is depicted in Figure 2. In the following, we briefly describe the four attacks.

2.6.1 Attack 1: Seq++ Attack

When an attacker receives an LSA instance, it modifies the link state metric and increments the LSA sequence number by 1 (i.e., Seq++). The attacker also needs to re-compute both the LSA and OSPF checksums before the tampered LSA instance is re-injected into the system. Because this attacking LSA has a larger sequence number, it will be considered “fresher” by other routers. Eventually it will be propagated to the originator of this particular LSA. The originator, according to the OSPF specification, will “fight back” with a new LSA carrying correct link status information and an even fresher sequence number.

The effect of this attack is an unstable network topology if the attacker keeps generating “Seq++” LSA instances. For example, all routers at one point will think the link cost is big (e.g., 100), but then the fight-back LSA instance from the originator will tell them the cost metric is much smaller (e.g., 10).

2.6.2 Attack 2: MaxAge Attack

When an attacker receives an LSA instance, it can modify the LSA age to *MaxAge* (i.e., 1 hour), and re-inject it into the system. This attacking LSA instance, with the same sequence number but age set to *MaxAge*, will cause all routers to purge the corresponding instance from their topology database. Eventually, the originator of this purged LSA will also receive this *MaxAge* LSA instance. The originator, according to the OSPF specification, will “fight-back” with a new

LSA instance carrying correct link status information and a fresher sequence number.

The effect of this attack is also an unstable network topology if the attacker keeps generating “MaxAge” LSA instances. For example, all routers at one point will think the link is not available, but then the fight-back LSA instance from the originator will tell them otherwise.

2.6.3 Attack 3: MaxSeq# Attack

When an attacker receives an LSA instance, it can modify the link state metric and set the LSA sequence number to 0x7FFFFFFF (i.e., MaxSequenceNumber). The attacker also needs to re-compute both the LSA and OSPF checksums before the tampered LSA instance is re-injected into the system. This attacking LSA instance, because it has the maximum LSA sequence number, will be considered the “freshest” by other routers. And, eventually it will be propagated to the originator of this particular LSA. The originator, according to the OSPF specification, “should” first purge the LSA instance (setting age equal to MaxAge) and then flood a new LSA carrying correct link status information and the smallest sequence number: 0x80000001.

We discovered in [9] that the effect of this attack depends on the implementation of the OSPF protocol. If the protocol is indeed implemented correctly, then it is similar to Seq++. On the other hand, many routing implementations do not handle the MaxSeq LSA correctly: the purging of the MaxSeq LSA is not implemented. We tested two different implementations, and neither of them implemented this purging mechanism. This implies that the MaxSeq LSA will stay in every router’s topology database for one hour before it reaches its MaxAge. In other words, an attacker can control the network topology database for upto one hour.

2.6.4 Attack 4: LSID Attack

According to the OSPF specification, the Link State ID and the Advertising router ID of a router LSA (type 1) should be the same. Some implementations did not check this requirement and take it for granted. When an attacker intercepts an outgoing type one LSA, it modifies its link state ID such that it’s different from the router ID. The victim (originator of this LSA) will use this LSID as a hash key to locate the database index pointer. If not found, a new pointer will be created. Later this pointer (NULL) is used to access the ls_age, which then causes a segmentation fault.

The effect of this attack is that the gated on the victim will stop running. Besides this serious damag-

ing effect, one interesting aspect of this attack is about the process of recovering this victim router. One has to stop the gated process on all of the routers in the same area before the victim can re-start its gated. It’s because the corrupted LSA is still sitting in the neighbors’ database. One has to first stop all the gated processes to effectively purge this bad LSA. Otherwise, when the victim restarts its gated, it will have a segmentation fault again by this bad LSA (received from one of its neighbors while it exchanges the database during start-up).

2.7 Testbed Configuration

In order to demonstrate attack effects and the detection capabilities of the JiNao IDS, a routing testbed has been setup with the configuration shown in Fig. 3. There are four subnets and six routers on the testbed. The routers are PCs running gated on either Linux or FreeBSD operating system. Traceroute is running on router 1 to reach router 6. The link costs are 5 for the links from router 2 to 172.16.121.X subnet and from router 4 to 172.16.127.X subnet. The rest of the links have a cost of 10. Based on this cost configuration, the traceroute reports the path of going from router 1 to 6 is 1-2-4-6.

The attacks are launched from router 3 to router 2 with the intention of modifying the link cost from 5 to 100 on router 2. If the attacks are successful, we should expect to see traceroute change its path to routers 3, 4, and 6. It was exactly what we observed after the attacks were launched.

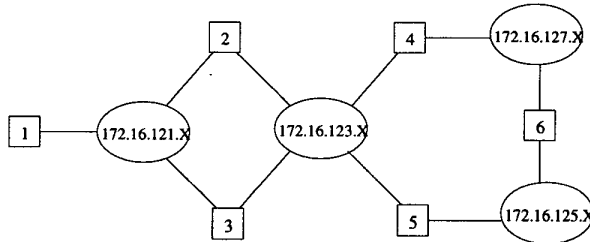


Figure 3: Routing testbed configuration

3 JiNao: Intrusion Detection System Architecture

In this section, we present an overview of JiNao’s system architecture design. The system consists of complementary functional blocks for providing comprehensive detection capabilities. It also incorporates standard network management functionalities to lay a foundation for facilitating automated responses in future research efforts.

Figure 4 illustrates the architecture design of our intrusion detection system. At the top level, there are two subsystems: namely, local detection subsystem and remote management subsystem. The remote management unit implements a set of network management applications which can both probe the status of and issue commands to the local detection subsystem. It was one of our design objectives that the JiNao system is capable of being integrated as part of an SNMP-based network management system.

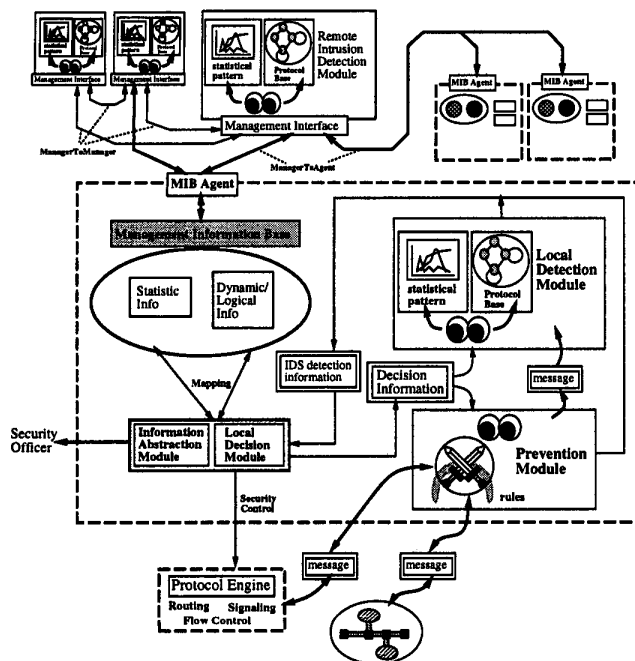


Figure 4: JiNao System Architecture.

3.1 Local Subsystem

A local subsystem consists of the following modules: rule-based prevention module, protocol and statistical-based detection modules, decision module, and local JiNao MIB agent.

3.1.1 Prevention Module

As the name "prevention" implies, this module will implement a small set of administrative/policy/firewall rules to filter out any packet with clear security violation before it enters into the router. The intent of the design is for this module to serve as a gate-keeper with a very short response time. The packets to be discarded include all those that may have significant damaging effect on the infrastructure according to general security guidelines or special security concerns of an administrative domain. The LSID

attack described in Section 2.6.4 is one clear example to justify the notion of attack prevention.

3.1.2 Detection Module

If a packet passes through the prevention module, it will be forwarded to the protocol engine for execution and to the local detection module which performs both statistical and protocol-based intrusion analysis. The results of these analyses are available for management application to access through SNMP. As shown in the Figure 4, the set of rules and their associated parameters in both prevention and detection modules can be dynamically modified by the remote management applications in response to the input of detection information. Therefore, our design allows certain degree of automated responses through the adoption of network management framework.

Statistical Analysis Module Intrusion detection using statistical analysis is founded on the contention that behavioral signatures exist for either users' usage profiles or protocol execution patterns (in this case, OSPF routing protocol) and intrusion will result in abnormal profiles. Any behavior deviating from the normal profile will be considered as an anomaly and appropriate alarms can be triggered. This statistical-based module provides the capability to detect intrusions that exploit previously unknown vulnerabilities. It is intended to uncover those attacks that cannot be prevented by a set of rules embedded in a rule-based component or cannot be detected by security analysis conducted through protocol-based approach.

Protocol Analysis Module The protocol-based approach detects intrusion by monitoring the execution of protocols in a router and triggering an intrusion alarms when an anomalous state is entered. Specifically, we have investigated the OSPF routing protocol operation through finite state machine (FSM) analysis.

3.1.3 Decision Module

The decision module serves as a coordinator to correlate the detection information from both the statistical and protocol analysis modules and takes appropriate action accordingly. One example to illustrate the function of the Decision Module is in the handling MaxSeq attack described in Section 2.6.3. According to the OSPF specification, 0x7ffffff is a legitimate sequence even though it takes at least 300 years for it to happen under a normal operation. One can choose to drop any LSA with MaxSeq right at the prevention module. However, a better way is to implement the modeling of MaxSeq attack in the protocol analysis module and turn on the wrapper function in the prevention module to handle the MaxSeq attack properly. (The wrapper

function processes the MaxSeq according to the OSPF specification.) The decision module can perform such coordination between the protocol analysis and prevention modules in the local sub-system.

3.1.4 Intrusion Detection MIB: JiNao MIB

As part of the network management framework, the SNMP MIB support in JiNao base (*JiNao MIB*) maintains and updates variables of interest. In the context of JiNao, the MIB is the database which stores the detection results from detection modules. The decision information issued from remote management applications can also be maintained through the MIB interface. The JiNao MIB specification has been defined and implemented.

3.2 Remote Subsystem

A remote subsystem consists of a set of management applications for monitoring and controlling a few local detection subsystems. A remote management application, for example, may sometimes re-configure the local detection system dynamically. With this configurability, the local detection subsystem can respond to intrusion differently under different situations.

4 JiNao Protocol Analysis Module

JiNao's Protocol Analysis Module (JPAM) utilizes the "knowledge" about the target protocol engine (e.g., OSPF), and examines the incoming and outgoing protocol engine traffic (e.g., OSPF traffic) to detect whether a known attack instance has been launched.

4.1 Architecture

JPAM consists of the following two modules as shown in Figure 5:

Event Abstraction Module: This module takes IP packets (e.g., OSPF protocol packets) as input and extracts/analyzes target protocol (e.g., OSPF) specific information. For instance, if the Decision module in JiNao is interested in LSA-level information for the LSA instances originated by a particular router *X*, then an Event Abstraction module instance will be created and configured to filter out those LSA instances that are related to this scope. The output of this module is a sequence of high-level events that will be consumed by one or more JFSM (JiNao Finite State Machine) pattern matching modules. In other words, each JFSM only handles LSA instances (originated from one particular router) representing the status information of the same link.

When an OSPF packet is intercepted from the router's kernel, JPAM will record the *timestamp* according to its local clock, and dispatch each LSA instance to appropriate Event Abstraction modules. If none of the JFSMs is interested in this LSA instance, it will be dropped immediately. Furthermore, it will decide whether it is an incoming (from other neighbor routers) or outgoing (from the router itself) LSA instance. The target Event Abstraction module will first analyze three fields in the LSA header: *Checksum*, *Age*, and *Seq#* to determine if any of the following four high-level events should be generated: "Invalid", "MaxAge", "MaxSeq", "MaxAgeSeq", or "SeqIncr". If none of the four events happens, this new LSA instance will be marked as a normal LSA update and input the event "UpdateLSA" to some JFSM pattern matching modules. According to the OSPF specification [5], in JiNao, we defined a set of LSA related events as partially shown in Table 1.

i.Invalid o.Invalid	indicates that LS checksum of the newly received LSA instance (incoming or outgoing, respectively) is invalid.
i.MaxAgeSeq o.MaxAgeSeq	indicates that the newly received LSA instance (incoming or outgoing, respectively) is with both <i>MaxAge</i> and <i>MaxSequenceNumber</i> , which is a purging LSA for the fightback purpose.
i.MaxSeq o.MaxSeq	indicates that the newly received LSA is with <i>MaxSequenceNumber</i> but without <i>MaxAge</i> .
i.InitSeq o.InitSeq	Incoming or Outgoing LSA with the minimal sequence number 0x80000001.
i.SeqIncr o.InitSeq	indicates that the router has just received or sent an LSA instance with a larger sequence number than the last LSA instance (but the same LSA) that was sent or received, respectively.
i.Update o.Update	indicates that the router has received or sent a regular Link-State Update packet with normal LSAs within the regular period.

Table 1: The Abstract Events for LSA: A Partial List

JFSM Pattern Matching Module: For each of such potential attacks, a pattern matching module instance will be created to process high-level events specially created for detecting that particular attack instance.

If a known attack is launched, at least one of the JFSM module instances should raise a real-time alarm event and report it to the decision module. On the other hand, if a JFSM instance can not handle the input event from current state, it will issue a message indicating the execution status of this instance. If all

the JFSM instances fail this way, it is likely that an unknown attack has occurred.

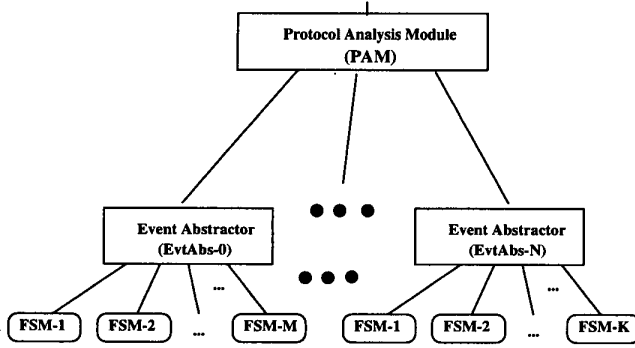


Figure 5: Architecture of JiNao Protocol Analysis Model

4.2 JiNao Real-Time FSM Specification

The JiNao Real-Time FSM (JFSM) model is a variant of the *I/O Automata* model defined in [4]. For modeling temporal relations among events, timing information is introduced in JFSM's state transition diagram. Therefore, in JFSM, a state transition will depend on not only the event identity itself but also the occurrence time of that event.

Definition 1 (JFSM M_i^{JFSM}) A JFSM, M_i^{JFSM} , consists of 9 tuples: $M_i^{JFSM} = (Q, \Sigma, \delta, q_0, F, FSMid, LSAid, ReportInfo, EffExtQue)$, where Q is a finite set of real-time states, Σ is the alphabet of events or input symbols, $\delta \subset Q \times \Sigma \times Q$ is the transition function mapping $Q \times \Sigma$ to Q . That is, $\Sigma(q, a)$ is a real-time state for each real-time state q and input symbol a . $q_0 \subseteq Q$ is the set of initial real-time states and $F \subseteq Q$ is the set of final real-time states. Furthermore, M_j also contains four more components related to intrusion detection: string *FSMId* to identify M_j , string *LSAid* to identify the LSA instances that M_j is handling, string *ReportInfo* to represent the message which will be sent out when intrusion detected (i.e., reaches a critical transition), and a queue *EffExtQue* for recording the effective execution.

Definition 2 (Input Events for M_i^{JFSM}) The input event e_i is a 3-tuple:

$$e_i = (ename, etime, edirection),$$

where *etime* is the timestamp for the occurrence of e_i , *ename* is the event identity as defined in Table 1. *edirection* is used to identify whether this is an "outgoing" or "incoming" LSA event.

Definition 3 (Real-Time State of M_i^{JFSM})

Each real-time state RtS_s in M_i^{JFSM} is a 5-tuple:

$$RtS_s = (sID, T_{in}, T_{last}, T_{current}, RtT),$$

where *sID* is RtS_s 's identity, T_{in} is the time when JFSM was entering RtS_s from another state RtS_p ($RtS_s \neq RtS_p$), T_{last} keeps the last time when a transition happened in the current state RtS_s (i.e., a transition loops back to the same state), $T_{current}$ is the current time, and RtT is the set of available real-time transitions for this state RtS_s .

According to the above definition, when M_i^{JFSM} is entering to a state RtS_s from another state RtS_p ($RtS_s \neq RtS_p$) because of the input event e_i , $RtS_s.T_{in}$ and $RtS_s.T_{current}$ will be updated to the timestamp of e_i , $e_i.etime$. $RtS_s.T_{last}$ will be always updated to $e_{(i-1).etime}$. If a new transition from a RtS_s to RtS_s itself, then $RtS_s.T_{in}$ will remain the same. That is, as long as M_i^{JFSM} stays in the current state (e.g., looping around the current state), $RtS_s.T_{in}$ will not be changed, only $RtS_s.T_{current}$ and $RtS_s.T_{last}$ will be updated. The elapsed time during which M_i^{JFSM} has stayed in the current state can be computed by $(RtS_s.T_{current} - RtS_s.T_{in})$. Furthermore, the inter-LSA arrival time is $(RtS_s.T_{current} - RtS_s.T_{last})$.

On the other hand, if a new transition is from a RtS_s to a different state RtS_n (i.e., $RtS_s \neq RtS_n$) because of e_j , $RtS_n.T_{in}$ will be updated to $e_j.etime$, while $RtS_s.T_{in}$ will still remain the same. Therefore, $(RtS_n.T_{in} - RtS_s.T_{in})$ represents how much time M_i^{JFSM} spent in RtS_s .

Whenever an input event occurs, M_i^{JFSM} takes that input and checks if there is a transition available for handling this input and determines if it can advance to next state or not. One single event may have more than one transitions and M_i^{JFSM} will process the first one which satisfies the given time constraints.

Definition 4 (Real-Time Transition of M_i^{JFSM})

RtT : Each real-time transition RtT_i in M_i^{JFSM} is a 8-tuple: $RtT_i = (RtS_{from}, e_{in}, \gamma, RtS_{to}, T_{min}^{state}, T_{max}^{state}, T_{min}^{interval}, T_{max}^{interval})$ where RtS_{from} is the "from" state, e_{in} can be either a real input event triggering RtT_i or a special NULL event e^ϕ , γ is the output result, RtS_{to} is the "to" state, T_{min}^{state} and T_{max}^{state} represent lower and upper bounds (integers in seconds), respectively, for the value of $(RtS_{to}.T_{in} - RtS_{from}.T_{in})$. In other words, for allowing RtT_i to occur, the time that M_i^{JFSM} has spent in RtS_{from} must be higher than T_{min}^{state} and lower than T_{max}^{state} . Similarly, $T_{min}^{interval}$ and $T_{max}^{interval}$ represent

lower and upper bounds (integers in seconds), respectively, for the value of $(RtS_{to}.T_{in} - RtS_{from}.T_{last})$. For allowing RtT_i to occur, the interval between two LSA events must be higher than $T_{min}^{interval}$ and lower than $T_{max}^{interval}$. Finally, RtT_i is effective if $RtT_i.RtS_{from} \neq RtT_i.RtS_{to}$.

M_i^{JFSM} may have two different transitions from the same state on the same input event. For example, we have:

$$RtT_1 = (RtS_{alice}, e_{Hello}, \gamma^1, RtS_{alice}, 0, 10, 0, \infty),$$

and,

$$RtT_2 = (RtS_{alice}, e_{Hello}, \gamma^2, RtS_{bob}, 11, \infty, 0, \infty).$$

If the e_{Hello} event occurs less than 11 seconds after M_i^{JFSM} entered RtS_{alice} , then RtT_1 will happen and JFSM will remain in RtS_{alice} . Otherwise, RtT_2 , a critical transition, will be triggered and M_i^{JFSM} will enter RtS_{bob} . Please note that a special case such as the following critical transition:

$$RtT_3 = (RtS_{alice}, e^\phi, \gamma^3, RtS_{bob}, 31, \infty, 0, \infty)$$

represents that, even if NO real events occur, M_i^{JFSM} will only stay in RtS_{alice} for at most 30 seconds.

Given an input string, an execution of M_i^{JFSM} can be represent as a sequence of transitions, (RtT_1, RtT_2, \dots) . The **effective execution** of M_i^{JFSM} consists of only those transitions that are **effective**. I.e., $\forall i, RtT_i.RtS_{from} \neq RtT_i.RtS_{to}$. In other words, an effective execution of M_i^{JFSM} only contains those transitions changing M_i^{JFSM} 's state.

Definition 5 (Critical Transition of M_i^{JFSM})

$RtT_f^{critical}$: A critical transition $RtT_f^{critical}$ is the last transition when M_i^{JFSM} detects an intrusion pattern. A critical transition will trigger M_i^{JFSM} to report a detection message to the decision module, which contains *FSMId*, *LSAid*, *ReportInfo*, and *EffExtQue*.

4.3 Detecting Known OSPF attacks: Example and Results

In our implementation of JiNao Protocol Analysis Module (JPAM), a JFSM instance is specified in a configuration file. To detect the attacks mentioned in section 2.6, we have developed a set of JFSM instance files. As an example, a JFSM to detect the Seq++ attack is depicted in Figures 6 and 7. In this example, JiNao is running on the **originator** of the monitored LSAs.

Lines 1-2 in Figure 7 describes two possible transitions from the initial state. Line 1 describes that, if the new **incoming** LSA's sequence number is bigger than the previous **outgoing** LSA (with the same *LSAid*), an *i.SeqIncr* event will trigger the transition and the JFSM will then be in State 1. In the attack scenario, this transition represents that the LSA's originator has received its own LSA instance with an abnormal sequence number – normally the sequence number should be the same as the one it just sent out.

However, this unusual event by itself is not enough to raise a red flag for the Seq++ attack. In OSPF, if a router crashed, its LSA instances will still be kept by other routers for about 30 minutes (or 1800 seconds). Therefore, when this router restarts within the 30 minutes limit, it can still receive "old" LSA instances with a bigger sequence number. Under such a case, the LSA's originator will issue another LSA with an even bigger LSA to "clean-up" the old copies out there. Line 3 represents this "clean-up" or "fight-back" scenario, and it is an *o.SeqIncr* as the originator will increment the LSA by at least one. Please note that at this state, the originator could receive more than one *i.SeqIncr* before the *o.SeqIncr* is out. The rationale is that, if the originator has more than one neighbors, it could receive more than one copy of *i.SeqIncr*. However, after the *o.SeqIncr* is out, it is impossible for an old copy of the LSA instance to raise an *i.SeqIncr* as the most recent outgoing sequence number has been updated. Finally, if the originator itself has crashed, then it will not perform the "fight-back" within the 30 minutes limit. Therefore, it will trigger the Line 4 transition and go back to the initial state.

Lines 6-8 in Figure 7 describes whether this originator receives another unusual *i.SeqIncr* from one of its neighbors after it delivered an *o.SeqIncr*. If this is indeed a Seq++ attack, the attacker will keep increasing the sequence number and therefore, in Line 6, within thirty minutes, we will receive another *i.SeqIncr*. The transition in Line 9 is **critical**, as after another fight-back from the originator is observed, the JFSM will raise a red alarm about Seq++ attack to the decision module.

Due to the space limit, we can not explain the details for all three attacks. In summary, we have developed three different JFSMs in handling those three attacks: Seq++, MaxAge, and MaxSeq. And, we have tested our JFSMs on three different routing testbed: MCNC, NCSU, and Air Force Rome Laboratory. Our experimental results demonstrated that all three attacks can be detected without any false positive.

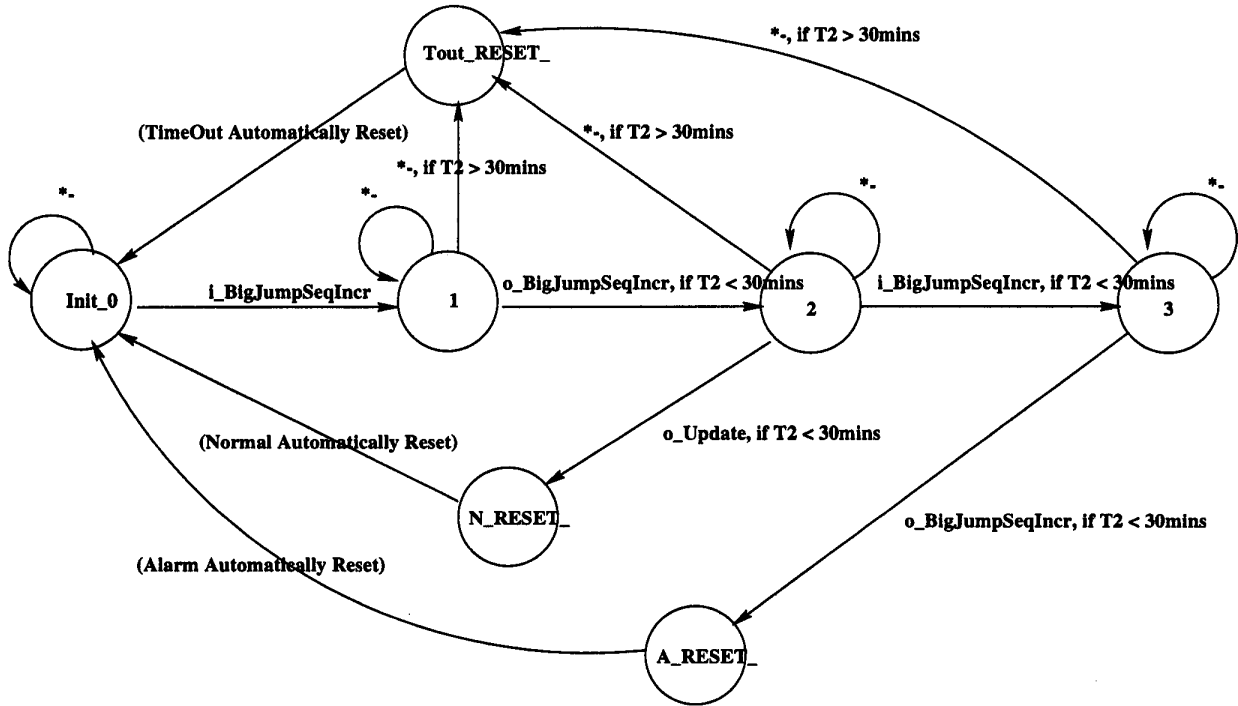


Figure 6: a JFSM diagram for LSA Seq++ Attack

FSMId: Seq++JFSM ReportInfo: For_Seq++
logToFileName: ./Seq++AttTest.log maxLogQueSize: 2000
Real-Time Transitions:

#	from	event	output	to	MinS	MaxS	MinI	MaxI	critical
#	---	---	---	---	---	---	---	---	---
01	0	i_SeqIncr	i_SeqIncr	1	0	inf	0	inf	NO
02	0	*	StayAt0	0	0	inf	0	inf	NO
03	1	o_SeqIncr	o_FightBack	2	0	1800	0	inf	NO
04	1	o_Update	NtFtBackOnTime	0	1800	inf	0	inf	NO
05	1	*	StayAt1	1	0	inf	0	inf	NO
06	2	i_SeqIncr	i_SeqIncrAgain	3	0	1800	0	inf	NO
07	2	o_Update	GoBkInit	0	0	inf	0	inf	NO
08	2	*	StayAt2	2	0	inf	0	inf	NO
09	3	o_SeqIncr	o_FightBackAtt	0	0	1800	0	inf	YES
10	3	*	GoBkInit	3	1800	inf	0	inf	NO
11	3	*	StayAt3	3	0	1800	0	inf	NO
#	---	---	---	---	---	---	---	---	---

Figure 7: a JFSM Configure File for LSA Seq++ Attack

5 JiNao Statistical Analysis Module (JSAM)

In the heart of JiNao's statistical analysis module is the NIDES/STAT algorithm [3] from SRI. The algorithm was adapted and implemented in detecting OSPF routing protocol attacks. This section first provides a brief discussion of the measure classification, both in the context of NIDES and JiNao. We then give a brief description of the NIDES algorithm, discuss its effectiveness when applying to OSPF routing protocol, and explain some fine-tunings we made to suit our needs.

5.1 Measure Classification

NIDES/STAT describes subject behavior by means of a profile, which is separated into short-term and long-term components. Aspects of a profile are represented as measures in the form of probability distribution. NIDES defined four classes of measures:

Activity Intensity Measures, which measure whether the volume of activity generated is normal; for instance, the OSPF packet volume generated by a certain application.

Categorical Measures, whose values are by nature categorical, like OSPF packet types, or files which were accessed.

Continuous or Counting Measures, whose values are numeric, like LSA age or CPU usage.

Audit Record Distribution, which monitors the distribution of all types of activity that have been generated in the recent past.

In terms of the probability distribution computation, each class of measures is done by the way of binning procedure for establishing histogram. Binning procedure refers to the computation in assigning a measure value to a correct bin in a histogram. Some measures have fixed bin end-points, like OSPF packet types (five types) and LSA age distribution (from 0 to 3600 seconds). Some measures, on the other hand, the bin end-points have to be determined first through other means. For instance, the upper bound of the OSPF packet volume has to be determined by collecting the mean and standard deviation of this intensity measure. (The upper bound is then set to be the mean plus five times of the standard deviation).

Due to the fact that binning procedure is used for each measure in NIDES/STAT's internal computation, we classified measures slightly differently in the JiNao project. There are only two classes of measures in JiNao: fixed measures and hash measures. Fixed

measures include those measures which have known (hence fixed) bin end-points (e.g., OSPF packet types and LSA age), while hash measures refer to those measures which bin end-points have to be derived (for instance, OSPF packet volume).

After the end points have been determined, the bins can be scaled either in a linear or geometric fashion. For fixed measures, the choice is most likely a linear scaling. The choice mainly depends on the applications and the data distribution for a hash measure. In the case of JiNao project, we found the linear scaling gave us a more even probability distribution of OSPF packet volume measure. It results in better stability and detection response. Therefore, linear scaling is adopted in JiNao for both classes of measures (and Q probability distribution, see Section 5.2.2).

5.2 NIDES/STAT Algorithm

NIDES/STAT algorithm monitors a subject's (either a user or a software program) behavior on a computer system, and raises alarm flags when the subject's current (short-term) behavior deviates significantly from its expected behavior, which is described by its long-term profile. NIDES's algorithm is based on a χ^2 -like test for comparing the similarity between the short-term and long-term profiles.

Some notations for describing the algorithm are in order. Let the current system behavior be a random variable under the sample space S . Events, E_1, E_2, \dots, E_k , represent a partition of S , where these k events are mutually exclusive and exhaustive. Let p_1, p_2, \dots, p_k be the probabilities of the occurrence corresponding to events E_1, E_2, \dots, E_k . The random experiment is repeated N times independently, where N is a large number. Furthermore, Y_i represents the number of occurrences for event E_i . Thus, we have $\sum_{i=1}^k (p_i) = 1$, where $p_i = Y_i/N$.

To examine whether a short-term profile has a similar probability distribution with the corresponding long-term profile, we test the following hypothesis:

$$H_0 : p'_i = p_i, i = 1, 2, \dots, k,$$

where $p'_i = Y'_i/N'$. Variables p'_i, Y'_i , and N' are associated with short-term profile. They denote the same meaning of their long-term profile counterparts.

Let

$$Q = \sum_{i=1}^k \frac{(Y'_i - N' \times p_i)^2}{N' \times p_i} \quad (1)$$

Intuitively, Q measures the "closeness" of the observed numbers to the corresponding expected numbers. A small Q favors hypothesis H_0 , while a large Q favors H_1 . If independence is assumed between events E_i ,

and the experiments are carried out independently, it has been proved that, for a large N , Q has an approximate χ^2 distribution with $k-1$ degree of freedom. To get an accurate approximation, it is suggested that N should be larger than 50 and $(N \times p_i)$ should be larger than 5. Otherwise, several “rare” events should be merged together to form a new event such that $(N \times p_{new})$ would exceed 5, where p_{new} denotes the probability for the new defined event.

Let q be an instance of Q . If $\Pr(Q > q) < \alpha$ (or $q > \chi^2_{\alpha}(k-1)$) where α is the desired significance level of the test, the hypotheses is rejected. In the context of our application, it means that the short-term profile is statistically different from its long-term profile which allows us to draw a conclusion that an anomalous behavior has occurred. Two kinds of errors are defined. *Type I error* means that the hypotheses is true but is rejected. *Type II error* means the Hypotheses is false, but is accepted. The probability that Type I error occurs is also refereed as *false positive rate*, while the probability for type II error is refereed as *false negative rate*.

In practice, however, the assumption of independence between events may not hold true. Therefore, Q may not have a χ^2 distribution. The NIDES/STAT algorithm proposed a way to track the values of Q in order to establish an empirical probability distribution for Q . This distribution, along with the distribution of the system’s expected behavior (i.e. p_1, p_2, \dots, p_k), is saved in a long-term profile, which is updated per UPDATE.PERIOD (24 hours in NIDES, 4 hours in our case) in a real-time operation. The details of operation is provided in Section 5.2.2.

Also the NIDES/STAT algorithm defines another variable S which is transformed from the tail probability of Q distribution such that S has a half-normal distribution. The purpose of defining S is to allow the degree of anomaly from different types of measures to be added on a comparable basis. The formula is:

$$S = \phi^{-1}\left(1 - \frac{\text{Prob}(Q > q)}{2}\right) \quad (2)$$

where ϕ is the cumulative normal distribution function of an $N(0, 1)$ variable, and again, $\text{Prob}(Q > q)$ is the tail probability.

5.2.1 Weighted Sum for Short-Term Profile

The NIDES/STAT algorithm compares system’s current short-term profile with its long-term profile. The sample size N dictates the time span of so called “short-term”. Intuitively, a “sliding window” should be implemented to keep the most recent N pieces of audit records. Whenever a new audit record arrives,

the window slides to cover it and the most remote (oldest) record is discarded. Then Y_i is updated, and Q and S are recalculated. But when N is big, the “sliding window” scheme would consume too much computing resources. NIDES/STAT algorithm proposed a *weighted sum* scheme to deal with this problem. A new variable N_{eff} , called effective N , is introduced. Each time an event E_i occurs, Y_i is recalculated with the following formula:

$$\begin{aligned} Y_i &= Y_i \times 2^{-\tau_s} + 1, \\ Y_j &= Y_j \times 2^{-\tau_s}, \quad j \neq i, \\ N_{eff} &= \sum_{i=1}^k (Y_i) = N_{eff} \times 2^{-\tau_s} + 1. \\ Count_i &= Count_i + 1 \end{aligned}$$

where τ_s is a pre-defined short-term fading factor, also referred to as *half-life* in NIDES/STAT algorithm. If τ_s is chosen to be $\frac{1}{100}$, the weight of a trail record will fade to 0.5 after 100 more records have been received. Note that N_{eff} no longer bears the original meaning of N which is the number of experiments. It is a weighted sum of all the experiments with the most recent one having the most weight. It can be easily derived that N_{eff} has an asymptotic value of $\frac{1}{(1-2^{-\tau_s})}$. Each category (event) has a counter, $Count_i$. It is used when long-term profile is updated, which is explained in next section.

Use this scheme, the calculations can be done recursively and efficiently. The computing time is proportional to k , which is the number of events. Typically it is much smaller than N , which could be several hundreds or even several thousands. Also this scheme saves lots of memory since it only requires to remember k variables rather than N records.

5.2.2 Long-Term Profile Training and Update

Training is the process by which the statistical component learns normal behavior for a subject. In NIDES/STAT, the profile training consists of three phases:

Category, C-Training: to learn the subject’s expected behavior, i.e., probabilities for events, E_i ;

Q Statistics, Q-Training: to learn empirical distribution for Q statistic which measures the deviation between short-term observations and long-term expected category distributions;

Threshold, T-Training: wherein the system establishes the threshold for the measures.

After training, the long-term profile is updated at a regular interval (for example, once per day in NIDES and every four hours in JiNao) to allow adaptation to gradual change of a subject's behavior. A long-term fading factor τ_l is defined so that the profile will "forget" the ancient data gradually. The details of the first two training phases are provided below. In our implementation, we did not adopt the T-Training due to certain practical concerns. The rationale is outlined in Section 5.2.3.

C-Training: Collect Expected Probabilities

After the high end-point of a hash measure is derived through a warm-up pre-C-Training phase, the following algorithm is applied to both hash and fixed measures to calculate the expected probabilities p_i . The calculation is activated at the end of each update period.

$$Upd_Period_Count = \sum_{i=1}^k Count_i,$$

$$H_{Neff} = H_{Neff} \times 2^{-\tau_l},$$

$$TempCount_i = H_{Neff} \times p_i + Count_i,$$

$$H_{Neff} = H_{Neff} + Upd_Period_Count,$$

$$p_i = TempCount_i / H_{Neff}$$

Upd_Period_Count has the total number of observations since the last long-term profile update across all categories (events) for a given measure. H_{Neff} is the historical effective number of observations. It is saved in long-term profile and aged by multiplying with 2 to the power $-\tau_l$. Now $H_{Neff} \times p_i$ gives aged count of the expected occurrence for category i (event E_i). Then p_i is updated by combining with the counts accumulated since the last update, $Count_i$. In *C-Training*, H_{Neff} has an initial value 0. While in regular update, H_{Neff} is retrieved from long-term profile.

Q-Training: Track Q 's Distribution To track the historical frequency distribution for Q , a procedure similar to the determination of the high end-point of a hash measure is adopted to derive the Q_{max} . A linear scale is adopted in the binning procedure of Q 's probability distribution.

Let Qp_i denote the probability with which Q is in the i^{th} bin. Each bin has an integer counter, $QCount_i$, which counts how many times Q falls into i^{th} bin. Assume q is an instance of Q , following formula is used to determine the number of bin which q is in :

$$i = \begin{cases} (q/Q_{max}) \times 31 & \text{if } q < Q_{max} \\ 31 & \text{if } q \geq Q_{max} \end{cases}$$

Then we simply increase bin i 's counter :

$$QCount_i = QCount_i + 1;$$

Qp_i is updated in the same way as p_i ,

$$Upd_Period_QCount = \sum_{i=0}^{31} QCount_i,$$

$$QH_{Neff} = QH_{Neff} \times 2^{-\tau_l},$$

$$QTempCount_i = QH_{Neff} \times Qp_i + QCount_i,$$

$$QH_{Neff} = QH_{Neff} + Upd_Period_QCount,$$

$$Qp_i = TempQCount_i / QH_{Neff}$$

Upd_Period_QCount has the total number that a measure has been activated. QH_{Neff} is the historical effective number. It is saved in long-term profile and aged by multiplying with 2 to the power $-\tau_l$ (yes, the same τ_l as in Section 5.2.2). $QH_{Neff} \times p_i$ gives aged count of the expected occurrence for bin i . Then Qp_i is updated by combining with the counts accumulated since the last update time, $QCount_i$. In *Q-Training* phase, QH_{Neff} is initialized to 0. While in regular update, QH_{Neff} is retrieved from long-term profile.

5.2.3 Score Anomalous Behavior

As each audit record is received, Q is computed according to equation (1) while changing N to N_{eff}

$$Q = \sum_{i=1}^k \frac{(Y_i - N_{eff} \times p_i)^2}{N_{eff} \times p_i}$$

Assume the result value is q . Then the tail probability is calculated as following:

$$Prob(Q > q) = \sum_{k=i}^{31} Qp_i$$

i is the bin index which q falls into.

As mentioned in Section 5.2, the tail probability $Prob(Q > q)$ is transformed into another variable S using formula (2). High S value corresponds to high q , thus corresponds to high degree of discrepancy between short-term and long-term profiles.

This is repeated for all measures, resulting in a vector of S values. As proposed in NIDES, all the S scores are combined into an overall statistics which is called $T2$. This statistic is a summary judgment of the abnormality of all active measures, and is given by the sum of the squares of the S statistics normalized by the number of measures:

$$T2 = \frac{\sum_{m=1}^{N_{measure}} S_m^2}{N_{measure}}$$

$N_{measure}$ is the total number of measures.

Similar to Q , the NIDES/STAT algorithm proposes to track the empirical distribution for $T2$. The distribution is collected during the last stage of the profile building period, after reasonably stable long-term distributions for the Q statistics have been constructed. A yellow alarm is raised when an instance of $T2$, say $t2$, is so high that $Prob(T2 > t2) < 0.01$, and a red alarm is raised when $Prob(T2 > t2) < 0.001$.

JiNao/STAT did not adopt $T2$ statistic because of following concerns:

1. It's hard for a security officer to understand the alarm. Due to the nature of statistical-base intrusion detection, it is possible that false alarms will be raised even without intrusion activity. So a security officer must investigate the reason for an alarm before he can tell if it is due to a intrusion or not. When configuring the IDS, he may choose several measures to monitor several different aspects of the system's behavior. We consider that it is better for each measure to raise alarm on its own. In this way, at least the officer can know which aspect(s) of the system went wrong.
2. This scheme may harm the sensibility of the IDS, and does not conform to the *pay more get more* principle. Different attacks may influence different aspects of the system's behavior. This is the reason one wants to choose several measures to guard against potential attacks. The effectiveness of the IDS depends on the assumption that it is hard for an attacker to attack a system without disturbing any of those monitored aspects. By adding the anomalies together, it is easier to detect those "careless" attacks which would disturb many aspects of system behavior, but the anomaly which is caused by "deliberate" attacks tends to be buried under other anomalies. Adding too many S s together makes it difficult for $T2$ to detect "deliberate" attacks.
3. Not all measures are activated for each audit records. For example, to protect an OSPF router, a measure is chosen to monitor the overall routing traffic a router receives, i.e. how many OSPF packets it has received during a period of time. Also, another measure is chosen to monitor the *LS age* field in every LSA. Now, assume the router receives an OSPF Hello packet (Hello packets are used to keep "adjacency" between neighboring routers. There is no LSA in Hello packet), the first measure is activated, while the second measure is skipped since there is no LSA at all.

Adding them together does not necessarily provide better attack indication.

Therefore, in JiNao/STAT, a measure is a stand-alone object. Each measure maintains its own state, and raises alarm on its own.

5.3 Experimental Results

The experimental results are represented as either **Red Alarm**, **Yellow Alarm**, or **Normal**, which represent the tail probability $Prob(Q > q)$ "< 0.1%", "between 0.1% and 1%", or "> 1%", respectively. The corresponding S statistic is 3.3 and 2.58 as the thresholds for red alarms and yellow alarms, respectively. In our implementation, the alarms were raised on average about 8-15 seconds after the attacks were launched on the testbed described in Section 2.7. All of the attacks would generate extra traffic with the exception of LSID attack. LSID attack will cause the gated on the victim to stop running and reduce the OSPF traffic. Statistical module can detect all the attacks with a low false positive rate.

6 Conclusions

In this paper, we presented the vulnerabilities of the OSPF link-state routing protocol and the JiNao intrusion detection system. We demonstrated in three real network testbeds: MCNC, NCSU, and AF/Rome Laboratory (mixture of PCs and commercial routers) that it is feasible to perform network infrastructure attacks to damage the network routing services. Through our analysis, if the target protocol itself is robust and self-stabilizing and is implemented correctly, the effect/damage to the network service is limited and easily detectable. For instance, the OSPF protocol will "fight-back" on any malicious attacks on the LSA (Link-State Advertisement) instances. If the attacker persistently launches these attacks, the attack instances can be detected very quickly and the source of the attacks can be identified.

While it's almost impossible to prevent these OSPF routing attacks, we showed that the JiNao IDS can effectively identify all three known OSPF/LSA attacks. In JiNao, the protocol analysis module is designed to catch known attacks, while the statistical analysis module is used to detect anomaly (unknown attacks). Our experiments show that the OSPF routing attacks can be detected by both modules with a low false positive. It is not surprising that the protocol analysis module performs very well as we use FSM with real-time extensions to model the behavior of those attacks. It is our observation that, compared to the long-term profiles collected under normal situations, the deviation caused by the attacks is very significant.

The JiNao IDS system can run on FreeBSD, Linux (real-time OSPF traffic, on-line), and Solaris (tcp-dump OSPF traffic, off-line). It has been integrated with the UCD-SNMP package so that a remote SNMP management application can manage JiNao through the SNMP JiNao-MIB interface.

Through the development of the JiNao system, we learned the following lessons:

Robust Protocol Design and Implementation: If a network protocol is not carefully designed, then it is extremely hard to develop an IDS to detect potential attacks. For instance, our experience says that *distance-vector (DV)* routing protocols are less detectable than *link-state (LS)* protocols. The key reason is that DV protocols will aggregate routing information along the route path, while LS protocols will flood the original link state information to all other routers in the same area.

Protocol Modulization and Event Abstraction: While studying only the protocol module, we found that many different types of low-level real-time events and states need to be considered. With the number of events and states, the FSMs for detecting the OSPF attacks can be very complicated. Thus, the development of the event abstraction module under JPAM is greatly useful in reducing the event space into 18 valuable high-level events. Based on these 18 events, JFSMs we developed contain only 3-4 states and 11-24 real-time transitions. As an example, the JFSM for detecting the Seq++ attack only contains 4 states and 11 transitions.

Selection of Statistical Measures: In our experiments about the statistical analysis module, we have used three different measures: "OSPF packet volume," "OSPF packet type" and "LSA age." "OSPF packet volume" is good for detecting Seq++ and MaxAge, while "OSPF packet type" is useful in dealing with Seq++ and MaxSeq. On the other hand, "LSA age" is only good for detecting MaxAge correctly. It is not clear to us whether there is a single statistical measure that can cover all three attacks. However, if we take only "OSPF packet type" and "LSA age", we are able to detect all the attacks with a very small false positive rate.

Efficient Implementation: Although the implementation of JiNao is in user space, the observed performance is acceptable. One key factor for achieving such good performance is that JiNao filters out unnecessary information or events in different network protocol layers in the kernel space.

References

- [1] Gregory G. Finn. Reducing the vulnerability of dynamic computer network. Technical report, Univ. of Southern California, ISI, June 1988.
- [2] Christian Huitema. *Routing in the Internet*. Prentice Hall, 1995.
- [3] Harold S. Javitz and Alfonso Valdest. The SRI IDES Statistical Anomaly Detector. In *Proc. of the IEEE Symposium on Research in security and Privacy*, pages 316-326, May 1991. <http://www2.csl.sri.com/nides/index5.html>.
- [4] Nancy Lynch and Mark Tuttle. An introduction to input/output automata. Technical report, Laboratory for Computer Science, Massachusetts Institute of Technology, September 1989. Technical Memo MIT/LCS/TM-373; <http://theory.lcs.mit.edu/tds/papers/Lynch/CWI89.html>.
- [5] J. Moy. RFC 2328: OSPF Version 2, April 1998. <ftp://ftp.isi.edu/in-notes/rfc2328.txt>.
- [6] S.L. Murphy and M.R. Badger. Digital signature protection of the ospf routing protocol. In *Proc. of the Internet Society Symposium on Network and Distributed Systems Security*, 1996.
- [7] B.R. Smith and J.J. Garcia-Luna-Aceves. Securing the Border Gateway Routing Protocol. In *Global Internet*, November 1996.
- [8] B.R. Smith, S. Murthy, and J.J. Garcia-Luna-Aceves. Securing Distance-Vector Routing Protocols. In *IEEE/ISOC Symposium on Network and Distributed System Security*, San Diego, CA, February 1997.
- [9] S.F. Wu B. Vetter, F. Wang. An experimental study of insider attacks for the ospf routing protocol. In *5th IEEE International Conference on Network Protocols, Atlanta, GA*. IEEE press, October 1997. <http://shang.csc.ncsu.edu/pubs.html>.
- [10] Divert Sockets for Linux. <http://www.anr.mcnc.org/~divert/>.
- [11] B. Vetter, F. Wang, and S.F. Wu. An Experimental Study of Insider Attacks for the OSPF Routing Protocol. In *IEEE International Conference on Network Protocols (ICNP)*, pages 293-300, October 1997.